

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 04-245543

(43)Date of publication of application : 02.09.1992

(51)Int.Cl.

G06F 9/42

(21)Application number : 03-244386

(71)Applicant : AMERICAN TELEPH & TELEGR CO <ATT>

(22)Date of filing : 29.08.1991

(72)Inventor : DEBRULER DENNIS L

(30)Priority

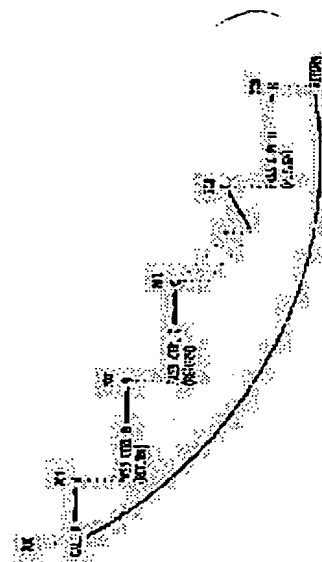
Priority number : 90 577427 Priority date : 04.09.1990 Priority country : US

(54) METHOD AND DEVICE FOR EXECUTING SUBPROGRAM ACCESS AND COMPILER DEVICE

(57)Abstract:

PURPOSE: To attain a high speed access by stopping the execution of an accessing subprogram in accordance with each access in a subprogram sequence from the accessing subprogram during the execution of the subprogram and starting the execution of an accessed subprogram.

CONSTITUTION: A subprogram accessing sequence for plural different subprograms 200 to 209 starts a 1st access, executes at least one intermediate access and executes a final access to end the operation. During the execution of the subprograms 200 to 209, the execution of an accessing subprogram is stopped in accordance with each access in the sequence by the accessing subprogram and the execution of an accessed subprogram is started. Consequently small memory consumption and high speed access to the subprograms 200 to 209 can be executed.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

Copyright (C); 1998,2003 Japan Patent Office

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平4-245543

(43) 公開日 平成4年(1992)9月2日

(51) Int.Cl.⁵

G 0 6 F 9/42

識別記号

3 3 0 A 9189-5B

庁内整理番号

F I

技術表示箇所

審査請求 未請求 請求項の数57(全 21 頁)

(21) 出願番号 特願平3-244386

(22) 出願日 平成3年(1991)8月29日

(31) 優先権主張番号 5 7 7 4 2 7

(32) 優先日 1990年9月4日

(33) 優先権主張国 米国 (US)

(71) 出願人 390035493

アメリカン テレフォン アンド テレグラフ
カムパニー

AMERICAN TELEPHONE
AND TELEGRAPH COMPA
NY

アメリカ合衆国、ニューヨーク、ニューヨ
ーク、マディソン アヴェニュー 550

(72) 発明者 デニス エル. デブルラー

アメリカ合衆国 60515 イリノイ、ダウ
ナズ グローブ メイン ストリート
4720

(74) 代理人 弁理士 三俣 弘文

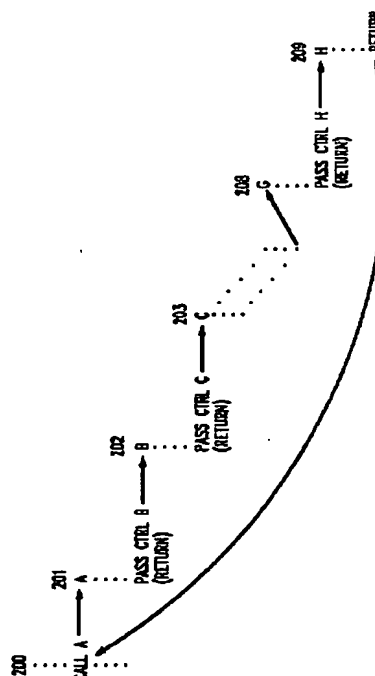
(54) 【発明の名称】 サブプログラム呼び出しを実行する方法および装置と

コンパイラ装置

(57) 【要約】

【目的】 サブプログラムの呼び出し連鎖において、メモリ消費の少ない高速な呼び出しを可能にする。

【構成】 PASS_CONTROL (図11) と呼ばれる配置が、従来のCALLおよびRETURNサブプログラム呼び出し系列 (図2) にとってかわるものとして、従来のRETURN文とともに使用される。本配置は、系列内の中間呼び出しを行ったサブプログラムへの中間的な復帰をすることなく、サブプログラム呼び出しの全系列から、系列を開始したサブプログラムへ直接復帰を実行する。本配置は、呼び出しの系列およびそこからの復帰を実行するために従来の実行スタック (114) を使用する (図12~14)。呼び出しの系列によって呼び出されたサブプログラムは実行スタック・フレーム (1620) を共有する。PASS_CONTROLの機能を実行するためのコンパイラ配置およびアプリケーション・プログラム実行配置が開示される。



1

【特許請求の範囲】

【請求項1】 複数の相異なるサブプログラムのサブプログラム呼び出しの系列を実行する方法において、前記系列が、第1呼び出しで開始し、続いて少なくとも1個の中間呼び出しがあり、最終呼び出しで終了するものであり、前記方法が、前記サブプログラム呼び出し系列内の呼び出しサブプログラムによるサブプログラムの各呼び出しに応じて、呼び出しサブプログラムの実行を停止し、呼び出されたサブプログラムの実行を開始するステップと、さらに前記系列内の第1呼び出しに応じて、前記第1呼び出しを行った呼び出しサブプログラムの実行を再開する命令へのポイントを実行スタック上に格納するステップと、前記系列内の最終呼び出しによって呼び出されたサブプログラムからの復帰に応じて、復帰元のサブプログラムの実行を停止し、前記実行スタックから前記命令ポイントを回復し、前記系列内の中間呼び出しを行ったサブプログラムの実行を中間で再開することなく、前記実行スタックから回復された命令ポイントによって指示される命令から、前記系列内の第1呼び出しを行ったサブプログラムの実行を再開するステップからなることを特徴とするサブプログラム呼び出しを実行する方法。

【請求項2】 前記系列内の第1呼び出しに続く各呼び出しに応じて、実行が再開される命令へのポイントを前記実行スタック上に格納することを抑制するステップからなることを特徴とする請求項1の方法。

【請求項3】 前記実行を再開するステップが、前記実行スタックから回復された命令ポイントによって指示される命令の実行に分岐するステップからなることを特徴とする請求項2の方法。

【請求項4】 前記命令ポイントを回復するステップが、前記実行スタックから、前記格納された命令ポイントを削除するステップからなることを特徴とする請求項3の方法。

【請求項5】 前記系列内の第1呼び出しに応じて、前記第1呼び出しを行った呼び出しサブプログラムの実行コンテキストを前記実行スタック上に格納するステップと、前記系列内の前記第1呼び出しに続く各呼び出しに応じて、コンテキストを前記実行スタック上に格納することを抑制するステップからなることを特徴とする請求項1の方法。

【請求項6】 前記系列内の最終呼び出しによって呼び出されたサブプログラムからの復帰に応じて、前記実行スタックから、実行コンテキストを、前記第1呼び出しを行ったサブプログラムの実行コンテキストに復元するステップからなることを特徴とする請求項5の方法。

【請求項7】 前記系列内の第1呼び出しに応じて、前記第1呼び出しを行った呼び出しサブプログラムの実行コンテキストを前記実行スタック上に格納するステップと、前記系列内の中間呼び出しを行ったサブプログラム

2

の実行コンテキストに前記実行コンテキストを中間で復元することなく、前記系列内の最終呼び出しによって呼び出されたサブプログラムからの復帰に応じて、前記実行スタックから、実行コンテキストを、前記第1呼び出しを行ったサブプログラムの実行コンテキストに復元するステップからなることを特徴とする請求項1の方法。

【請求項8】 前記ポイントを格納するステップが、前記系列内の第1呼び出しに応じて、前記実行スタック上に前記呼び出されたサブプログラムのスタック・フレームを作成するステップと、前記スタック・フレーム内に前記命令ポイントを格納するステップからなり、前記方法がさらに、前記系列内の第1呼び出しに続く各呼び出しに応じて、前記呼び出されたサブプログラムのための新たなスタック・フレームの作成を抑制するステップからなり、前記命令ポイントを回復するステップが、前記実行スタックから前記スタック・フレームを削除するステップを含むことを特徴とする請求項1の方法。

【請求項9】 前記系列内の第1呼び出しに続く各呼び出しに応じて、実行が再開される命令へのポイントを前記実行スタック上に格納することを抑制するステップからなることを特徴とする請求項8の方法。

【請求項10】 前記系列内の第1呼び出しに応じて、前記第1呼び出しを行った呼び出しサブプログラムの実行コンテキストを前記スタック・フレームに格納するステップと、前記系列内の前記第1呼び出しに続く各呼び出しに応じて、コンテキストを前記実行スタック上に格納することを抑制するステップからなることを特徴とする請求項8の方法。

【請求項11】 前記系列内の最終呼び出しによって呼び出されたサブプログラムからの復帰に応じて、前記スタック・フレームから、実行コンテキストを、前記第1呼び出しを行ったサブプログラムの実行コンテキストに復元するステップからなることを特徴とする請求項10の方法。

【請求項12】 前記ポイントを格納するステップが、前記系列内の第1呼び出しに応じて、前記実行スタック上に前記呼び出されたサブプログラムのスタック・フレームを作成するステップと、前記スタック・フレーム内に前記命令ポイントを格納するステップと、前記呼び出されたサブプログラムの情報を前記スタック上に格納するステップからなり、前記方法がさらに、前記系列内の第1呼び出しに続く各呼び出しに応じて、前記呼び出されたサブプログラムのための新たなスタック・フレームを作成することなく前記呼び出されたサブプログラムの情報を前記スタックに格納するステップからなり、前記命令ポイントを回復するステップが、前記実行スタックから前記スタック・フレームをすべて削除するステップと、前記スタック・フレームに格納されたポイントによって指示される命令の実行に分岐するステップを含むことを特徴とする請求項1の方法。

3

【請求項13】 前記系列内の第1呼び出しに続く各呼び出しに応じて、実行が再開される命令へのポインタを前記実行スタック上に格納することを抑制するステップからなることを特徴とする請求項12の方法。

【請求項14】 前記系列内の第1呼び出しに応じて、前記第1呼び出しを行った呼び出しサブプログラムの実行コンテキストを前記スタック・フレームに格納するステップと、前記系列内の前記第1呼び出しに続く各呼び出しに応じて、コンテキストを前記実行スタック上に格納することを抑制するステップと、前記系列内の最終呼び出しによって呼び出されたサブプログラムからの復帰に応じて、前記スタック・フレームから、実行コンテキストを、前記第1呼び出しを行ったサブプログラムの実行コンテキストに復元するステップからなることを特徴とする請求項13の方法。

【請求項15】 前記呼び出されたサブプログラムの情報を前記スタック上に格納する各ステップが、前記呼び出されたサブプログラムの情報のための領域を前記スタック・フレーム内に確保するステップと、前記呼び出されたサブプログラムのために確保された領域に前記呼び出されたサブプログラムの情報を格納するステップからなることを特徴とする請求項14の方法。

【請求項16】 前記呼び出されたサブプログラムの情報を前記スタック上に格納するステップがそれぞれ、前記実行スタック上のスタック・フレームに先行するパラメータ領域に、前記呼び出しサブプログラムによって前記呼び出されたサブプログラムに渡されたパラメータを格納するステップからなることを特徴とする請求項15の方法。

【請求項17】 前記第1呼び出しに応じて前記呼び出されたサブプログラムの情報を前記スタック上に格納するステップが、呼び出されたすべてのサブプログラムの情報に対して単一の領域を前記スタック・フレーム上に確保するステップからなり、前記呼び出されたサブプログラムの情報を前記スタック上に格納する各ステップが、前記呼び出されたサブプログラムの情報を前記1個の確保された領域に格納するステップをからなることを特徴とする請求項14の方法。

【請求項18】 前記呼び出されたサブプログラムの情報を前記スタック上に格納するステップがそれぞれ、前記実行スタック上のスタック・フレームに先行するパラメータ領域に、前記呼び出しサブプログラムによって前記呼び出されたサブプログラムに渡されたパラメータを格納するステップからなることを特徴とする請求項17の方法。

【請求項19】 複数の相異なるサブプログラムのサブプログラム呼び出しの系列を実行する方法において、前記系列が、第1呼び出しで開始し、続いて少なくとも1個の中間呼び出しがあり、最終呼び出しで終了するものであり、前記方法が、前記系列内の第1呼び出しに

4

て、呼び出されるサブプログラムをCALLするステップと（このステップは、前記第1呼び出しを行った呼び出しサブプログラムの実行を再開する命令へのポインタを実行スタック上に格納するステップを含む）、前記系列内の後続の各呼び出しに応じて、前記呼び出されたサブプログラムにPASS_CONTROLするステップと（このステップは、実行が再開される命令へのポインタを格納することを抑制するステップを含む）、前記系列内の中間呼び出しを行ったサブプログラムの実行を中間で再開することなく、前記系列内の最終呼び出しによって呼び出されたサブプログラムからの復帰に応じて、前記系列内の第1呼び出しを行ったサブプログラムの実行に、前記実行スタック上に格納された命令ポインタにおいて直接RETURNするステップからなることを特徴とするサブプログラム呼び出しを実行する方法。

【請求項20】 複数の相異なるサブプログラムのサブプログラム呼び出しの系列を実行する装置において、前記系列が、第1呼び出しで開始し、続いて少なくとも1個の中間呼び出しがあり、最終呼び出しで終了するものであり、前記装置が、複数の相異なるサブプログラムを実行する手段からなり、前記サブプログラム呼び出し系列内の呼び出しサブプログラムによるサブプログラムの各呼び出しに応じて、呼び出しサブプログラムの実行を停止し、呼び出されたサブプログラムの実行を開始する第1手段と、さらに前記系列内の第1呼び出しに応じて、前記第1呼び出しを行った呼び出しサブプログラムの実行を再開する命令へのポインタを実行スタック上に格納する第2手段と、前記系列内の最終呼び出しによって呼び出されたサブプログラムからの復帰に応じて、復帰元のサブプログラムの実行を停止し、前記実行スタックから前記命令ポインタを回復し、前記系列内の中間呼び出しを行ったサブプログラムの実行を中間で再開することなく、前記実行スタックから回復された命令ポインタによって指示される命令から、前記系列内の第1呼び出しを行ったサブプログラムの実行を再開する第3手段からなることを特徴とするサブプログラム呼び出しを実行する装置。

【請求項21】 前記系列内の第1呼び出しに続く各呼び出しに応じて、実行が再開される命令へのポインタを前記実行スタック上に格納することを抑制する手段からなることを特徴とする請求項20の装置。

【請求項22】 前記第3手段がさらに、前記実行スタックから回復された命令ポインタによって指示される命令の実行に分岐することによって前記復帰に対応することを特徴とする請求項21の装置。

【請求項23】 前記第3手段がさらに、前記実行スタックから、前記格納された命令ポインタを削除することによって前記復帰に対応することを特徴とする請求項22の装置。

【請求項24】 前記第2手段がさらに、前記系列内の

5

第1呼び出しに応じて、前記第1呼び出しを行った呼び出しサブプログラムの実行コンテキストを前記実行スタック上に格納し、前記サブプログラムの実行手段が、前記系列内の前記第1呼び出しに続く各呼び出しに応じて、コンテキストを前記実行スタック上に格納することを抑制することを特徴とする請求項20の装置。

【請求項25】 前記第3手段がさらに、前記実行スタックから、実行コンテキストを、前記第1呼び出しを行ったサブプログラムの実行コンテキストに復元することによって、前記復帰に対応することを特徴とする請求項24の装置。

【請求項26】 前記第2手段がさらに、第1呼び出しに応じて、前記第1呼び出しを行った呼び出しサブプログラムの実行コンテキストを前記実行スタック上に格納し、前記第3手段がさらに、前記系列内の中間呼び出しを行ったサブプログラムの実行コンテキストに前記実行コンテキストを途中で復元することなく、前記復帰に応じて、前記実行スタックから、実行コンテキストを、前記第1呼び出しを行ったサブプログラムの実行コンテキストに復元することを特徴とする請求項20の装置。

【請求項27】 前記第2手段が、前記第1呼び出しに応じて、前記実行スタック上に前記呼び出されたサブプログラムのスタック・フレームを作成する手段と、前記第1呼び出しに応じて、前記スタック・フレーム内に前記命令ポインタを格納する手段からなり、前記サブプログラムを実行する手段が、前記系列内の第1呼び出しに続く各呼び出しに応じて、前記呼び出されたサブプログラムのための新たなスタック・フレームの作成を抑制し、前記第3手段が、前記復帰に応じて、前記実行スタックから前記スタック・フレームを削除する手段を含むことを特徴とする請求項20の装置。

【請求項28】 前記サブプログラムを実行する手段が、前記系列内の第1呼び出しに続く各呼び出しに応じて、実行が再開される命令へのポインタを前記実行スタック上に格納することを抑制することを特徴とする請求項27の装置。

【請求項29】 前記第2手段がさらに、前記第1呼び出しに応じて、前記第1呼び出しを行った呼び出しサブプログラムの実行コンテキストを前記スタック・フレームに格納する手段と、前記サブプログラムを実行する手段が、前記系列内の前記第1呼び出しに続く各呼び出しに応じて、コンテキストを前記実行スタック上に格納することを抑制することを特徴とする請求項27の装置。

【請求項30】 前記第3手段が、前記復帰に応じて、前記スタック・フレームから、実行コンテキストを、前記第1呼び出しを行ったサブプログラムの実行コンテキストに復元する手段を含むことを特徴とする請求項29の装置。

【請求項31】 前記第2手段が、前記第1呼び出しに応じて、前記実行スタック上に前記呼び出されたサブ

6

プログラムのスタック・フレームを作成する手段と、前記第1呼び出しに応じて、前記スタック・フレーム内に前記命令ポインタを格納する手段を含み、前記第1手段が、前記系列内の各呼び出しに応じて、前記呼び出されたサブプログラムの情報を前記スタックに格納する第4手段からなり、前記サブプログラムを実行する手段が、前記系列内の第1呼び出しに続く各呼び出しに応じて、呼び出されたサブプログラムのための新たなスタック・フレームを作成することを抑制し、前記第3手段が、前記復帰に応じて、前記実行スタックから前記スタック・フレームをすべて削除する手段と、前記復帰に応じて、前記スタック・フレームに格納されたポインタによって指示される命令の実行に分岐する手段を含むことを特徴とする請求項20の装置。

【請求項32】 前記サブプログラムを実行する手段が、前記系列内の第1呼び出しに続く各呼び出しに応じて、実行が再開される命令へのポインタを前記実行スタック上に格納することを抑制することを特徴とする請求項31の装置。

【請求項33】 前記第2手段がさらに、前記第1呼び出しに応じて、前記第1呼び出しを行った呼び出しサブプログラムの実行コンテキストを前記スタック・フレームに格納する手段を含み、前記サブプログラムを実行する手段が、前記系列内の前記第1呼び出しに続く各呼び出しに応じて、コンテキストを前記実行スタック上に格納することを抑制し、前記第3手段が、前記復帰に応じて、前記スタック・フレームから、実行コンテキストを、前記第1呼び出しを行ったサブプログラムの実行コンテキストに復元する手段を含むことを特徴とする請求項31の装置。

【請求項34】 前記第4手段が、前記呼び出されたサブプログラムの情報のための領域を前記スタック・フレーム内に確保する手段と、前記呼び出されたサブプログラムのために確保された領域に前記呼び出されたサブプログラムの情報を格納する手段からなることを特徴とする請求項33の装置。

【請求項35】 前記第4手段が、前記実行スタック上のスタック・フレームに先行するパラメータ領域に、前記呼び出しサブプログラムによって前記呼び出されたサブプログラムに渡されたパラメータを格納する手段からなることを特徴とする請求項34の装置。

【請求項36】 前記第4手段が、前記第1呼び出しに応じて、呼び出されたすべてのサブプログラムの情報に対して単一の領域を前記スタック・フレーム上に確保する手段と、前記系列内の各呼び出しに応じて、前記呼び出されたサブプログラムの情報を前記1個の確保された領域に格納する手段からなることを特徴とする請求項33の装置。

【請求項37】 前記第4手段が、前記実行スタック上のスタック・フレームに先行するパラメータ領域に、前

7

記呼び出しサブプログラムによって前記呼び出されたサブプログラムに渡されたパラメータを格納する手段からなることを特徴とする請求項36の装置。

【請求項38】 複数の相異なるサブプログラムのサブプログラム呼び出しの系列を実行する装置において、前記系列が、第1呼び出しで開始し、続いて少なくとも1個の中間呼び出しがあり、最終呼び出しで終了するものであり、前記装置が、前記系列内の第1呼び出しに応じて、呼び出されるサブプログラムをCALLする第1手段と（この手段は、前記第1呼び出しを行った呼び出しサブプログラムの実行を再開する命令へのポインタを実行スタック上に格納することを含む）、前記系列内の後続の各呼び出しに応じて、前記呼び出されたサブプログラムにPASS_CONTROLする第2手段と（この手段は、実行が再開される命令へのポインタを格納することを抑制することを含む）、前記系列内の中間呼び出しを行ったサブプログラムの実行を途中で再開することなく、前記系列内の最終呼び出しによって呼び出されたサブプログラムからの復帰に応じて、前記系列内の第1呼び出しを行ったサブプログラムの実行に、前記実行スタック上に格納された命令ポインタにおいて直接RETURNする第3手段からなることを特徴とするサブプログラム呼び出しを実行する装置。

【請求項39】 サブプログラム呼び出し文およびサブプログラム復帰文を含むソース・コード文から構成される複数の相異なるソース・サブプログラムからなるソース・プログラムを、オブジェクト命令から構成される複数のオブジェクト・サブプログラムからなるオブジェクト・プログラムへとコンパイルする手段からなるコンパイラ装置において、前記手段が、ソース・サブプログラムにおいてソース・サブプログラムを呼び出す文に遭遇することに応じて、前記呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行から、前記呼び出されたソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行へ分岐する命令を生成する第1手段と、ソース・サブプログラムにおいてソース・サブプログラムを呼び出す文に遭遇することに応じて、前記文は複数の相異なるソース・サブプログラムの呼び出し文の系列内の第1呼び出し文であり、前記系列内の前記第1呼び出し文を除く文はそれぞれ前記系列内の先行する呼び出し文によって呼び出されたソース・サブプログラムに含まれ、前記系列は前記第1呼び出し文で開始し、少なくとも1個の中間呼び出し文が続き、最終呼び出し文で終了するものであり、前記第1呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行を再開する命令へのポインタを実行スタック上に格納する命令を生成する第2手段と、前記系列内の最終呼び出し文によって呼び出されたソース・サブプログラム内の復帰文に遭遇することに応じて、前記

8

実行スタックから前記命令ポインタを回復し、前記系列内の中間呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行を途中で再開することなく、前記復帰文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行から、前記系列の第1呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行へ、前記実行スタックから回復された命令ポインタによって指示される命令において、直接分岐する第3手段からなることを特徴とするコンパイラ装置。

【請求項40】 前記ソース・プログラムをコンパイルする手段が、ソース・プログラムにおいて前記系列内の第1呼び出し文に続く各呼び出し文に遭遇することに応じて、実行が再開される命令へのポインタを前記実行スタック上に格納する命令の生成を抑制することを特徴とする請求項39の装置。

【請求項41】 前記第3手段がさらに、前記復帰文に遭遇することに応じて、前記格納された命令ポインタを前記実行スタックから削除する命令を生成することを特徴とする請求項40の装置。

【請求項42】 前記第2手段がさらに、前記系列内の第1呼び出し文に遭遇することに応じて、前記第1呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行コンテキストを前記実行スタック上に格納する命令を生成し、前記ソース・プログラムをコンパイルする手段が、前記系列内の第1呼び出し文に続く各呼び出し文に遭遇することに応じて、コンテキストを前記実行スタック上に格納する命令の生成を抑制することを特徴とする請求項39の装置。

【請求項43】 前記第3手段がさらに、前記復帰文に遭遇することに応じて、前記実行スタックから、前記第1呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行コンテキストへ、実行コンテキストを復元する命令を生成することを特徴とする請求項42の装置。

【請求項44】 前記第2手段がさらに、前記第1呼び出し文に遭遇することに応じて、前記第1呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行コンテキストを前記実行スタック上に格納する命令を生成し、前記第3手段がさらに、前記復帰文に遭遇することに応じて、前記系列内の中間呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行コンテキストへ、前記実行コンテキストを途中で復元する命令を生成することなく、前記実行スタックから、前記第1呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行コンテキストへ、実行コンテキストを復元する命令を生成

することを特徴とする請求項39の装置。

【請求項45】 前記第2手段が、前記第1呼び出し文に遭遇することに応じて、前記実行スタック上に前記呼び出されたサブプログラムのスタック・フレームを作成する命令を生成する手段と、前記第1呼び出し文に遭遇することに応じて、前記スタック・フレーム内に前記命令ポインタを格納する命令を生成する手段からなり、前記ソース・プログラムをコンパイルする手段が、前記系列内の第1呼び出し文に続く各呼び出し文に遭遇することに応じて、前記呼び出されたサブプログラムのための新たなスタック・フレームを作成する命令の生成を抑制し、前記第3手段が、前記復帰文に遭遇することに応じて、前記実行スタックから前記スタック・フレームを削除する命令を生成する手段を含むことを特徴とする請求項39の装置。

【請求項46】 前記ソース・プログラムをコンパイルする手段が、前記系列内の第1呼び出し文に続く各呼び出し文に遭遇することに応じて、実行が再開される命令へのポインタを前記実行スタック上に格納する命令の生成を抑制することを特徴とする請求項45の装置。

【請求項47】 前記第2手段がさらに、前記第1呼び出し文に遭遇することに応じて、前記第1呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行コンテキストを前記スタック・フレームに格納する命令を生成する手段と、前記ソース・プログラムをコンパイルする手段が、前記系列内の前記第1呼び出し文に続く各呼び出し文に遭遇することに応じて、コンテキストを前記実行スタック上に格納する命令の生成を抑制することを特徴とする請求項45の装置。

【請求項48】 前記第3手段が、前記復帰文に遭遇することに応じて、前記スタック・フレームから、実行コンテキストを、前記第1呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行コンテキストに還元する命令を生成する手段を含むことを特徴とする請求項47の装置。

【請求項49】 前記第2手段が、前記第1呼び出し文に遭遇することに応じて、前記実行スタック上に、前記呼び出されたソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムのスタック・フレームを作成する命令を生成する手段と、前記第1呼び出し文に遭遇することに応じて、前記スタック・フレーム内に前記命令ポインタを格納する命令を生成する手段を含み、前記第1手段が、前記系列内の各呼び出し文に遭遇することに応じて、前記呼び出されたソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの情報を前記スタックに格納する命令を生成する第4手段からなり、前記ソース・プログラムをコンパイルする手段が、前記系列内の第1呼び出し文に続く各呼び出し文に遭遇することに応じて、呼び出されたソース・

サブプログラムからコンパイルされたオブジェクト・サブプログラムのための新たなスタック・フレームを作成する命令の生成を抑制することを特徴とする請求項39の装置。

【請求項50】 前記ソース・プログラムをコンパイルする手段が、前記系列内の第1呼び出し文に続く各呼び出し文に遭遇することに応じて、実行が再開される命令へのポインタを前記実行スタック上に格納する命令の生成を抑制することを特徴とする請求項49の装置。

【請求項51】 前記第2手段がさらに、前記第1呼び出し文に遭遇することに応じて、前記第1呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行コンテキストを前記スタック・フレームに格納する命令を生成する手段を含み、前記ソース・プログラムをコンパイルする手段が、前記系列内の前記第1呼び出し文に続く各呼び出し文に遭遇することに応じて、コンテキストを前記実行スタック上に格納する命令の生成を抑制し、前記第3手段が、前記復帰文に遭遇することに応じて、前記スタック・フレームから、実行コンテキストを、前記第1呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行コンテキストに還元する命令を生成する手段を含むことを特徴とする請求項49の装置。

【請求項52】 前記第4手段が、前記呼び出されたソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの情報のための領域を前記スタック・フレーム内に確保する命令を生成する手段と、前記呼び出されたソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムのために確保された領域に、前記呼び出されたサブプログラムからコンパイルされたオブジェクト・サブプログラムの情報を格納する命令を生成する手段からなることを特徴とする請求項51の装置。

【請求項53】 前記第4手段が、前記実行スタック上のスタック・フレームに先行するパラメータ領域に、前記呼び出しソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムによって、前記呼び出されたソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムに渡されたパラメータを格納する命令を生成する手段からなることを特徴とする請求項52の装置。

【請求項54】 前記第4手段が、前記第1呼び出し文に遭遇することに応じて、呼び出し文によって呼び出されたソース・サブプログラムからコンパイルされたすべてのオブジェクト・サブプログラムの情報に対して単一の領域を前記スタック・フレーム上に確保する命令を生成する手段と、前記系列内の各呼び出し文に遭遇することに応じて、前記呼び出されたソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの

情報を前記1個の確保された領域に格納する命令を生成する手段からなることを特徴とする請求項51の装置。

【請求項55】 前記第4手段が、前記実行スタック上のスタック・フレームに先行するパラメータ領域に、前記呼び出しソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムによって、前記呼び出されたソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムに渡されたパラメータを格納する命令を生成する手段からなることを特徴とする請求項54の装置。

【請求項56】 前記コンパイル手段が、格納プログラムの制御下で動作する汎用データ・プロセッサを含み、前記格納プログラムは、前記データ・プロセッサが、前記ソース・プログラムを前記オブジェクト・プログラムへとコンパイルすることを可能にする命令のセットを含むことを特徴とする請求項39の装置。

【請求項57】 サブプログラム呼び出し文およびサブプログラム復帰文を含むソース・コード文から構成される複数の相異なるソース・サブプログラムからなるソース・プログラムを、オブジェクト命令から構成される複数のオブジェクト・サブプログラムからなるオブジェクト・プログラムへとコンパイルする装置が、ソース・サブプログラムにおいてソース・サブプログラムをCALLする文に遭遇することに応じて、前記文は複数の相異なるサブプログラムの呼び出し文の系列内の第1呼び出し文であり、前記系列内の前記第1呼び出し文を除く文はそれぞれ前記系列内の先行する呼び出し文によって呼び出されたソース・サブプログラムに含まれ、前記系列は前記第1呼び出し文で開始し、少なくとも1個の中間呼び出し文が続き、最終呼び出し文で終了するものであり、(a) 前記第1呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行を再開する命令へのポインタを実行スタック上に格納する命令と、(b) 前記呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行から、前記呼び出されたソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行へ分岐する命令を生成する第1手段と、ソース・サブプログラムにおいて前記系列内の後続の各呼び出し文に遭遇することに応じて、前記後続文がそれぞれソース・サブプログラムにPASS_CONTROLし、(a) 前記呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行から、前記呼び出されたソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行へ分岐する命令を生成し、(b) 実行が開始される命令へのポインタを格納する命令の生成を抑制する第2手段と、前記系列内の最終呼び出し文によって呼び出されたソース・サブプログラム内のRETURN文に遭遇することに応じて、前記実行スタックから前記

命令ポインタを回復し、前記系列内の中間呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行を途中で再開することなく、前記RETURN文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行から、前記系列の第1呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行へ、前記実行スタックから回復された命令ポインタによって指示される命令において、直接分岐する第3手段からなることを特徴とするコンパイラ装置。

【発明の詳細な説明】

【0001】

【産業上の利用分野】 本発明は、コンピュータにおけるサブプログラムの実行を連結するための装置に関する。

【0002】

【従来の技術】 サブプログラムは、他のプログラムの実行を呼び出し、呼び出されたプログラムから最終的にはその実行が復帰されるようなプログラムであるか、または、その実行が他のプログラムから呼び出され、それを呼び出したプログラムのような他のプログラムに実行を復帰することによって終了するようなプログラムである。もちろん、プログラムは、呼び出し側のサブプログラムであることも、呼び出される側のサブプログラムであることも可能である。サブプログラムは、ルーチン、サブルーチン、ファンクション、手続きのような、他の名前でも知られている。ここで使用される「サブプログラム」という語はこのようなすべての変化を包含する。

【0003】 サブプログラムは、構造化、またはモジュール化プログラミングにおいて広範に使用されている。このプログラミング法は、1個のタスクを一連のサブタスクに分解する、一般的に使用されているプログラミング技術である。構造化プログラミングはサブプログラムの長い連鎖を生じることがある。この場合、初期、またはメイン・サブプログラムがサブプログラムAを呼び出し、続いてこれがサブプログラムBを呼び出し、さらにこれがサブプログラムCを呼び出す、などとなる。

【0004】 このようなサブプログラムの連鎖は、プロトコル・ハンドラ、トランザクション・ハンドラ、データベース照会サーバ、通話処理システムのようなある種のアプリケーション・プログラムに特徴的である。この種のアプリケーション・プログラムはまたスループット制約条件によっても特徴づけられる。他のすべてのことが等しい場合、スループットはプログラム実行速度に直接比例する。従って、サブプログラムの呼び出しおよび呼び出しからの復帰はできるだけ迅速に実行可能であることが重要である。

【0005】 もちろん、プログラム実行を高速化する1つの方法は、より高速なコンピュータを使用することである。しかし、コンピュータ速度は一般的にコンピュー

13

タ・コストに直接比例するため、この方法は高価である。さらに、技術開発は絶えず続いているので、技術は実現可能なプログラム実行速度に実際的な制限を設定せざるを得ない。従って、サブプログラムの呼び出しおよび呼び出しからの復帰は、与えられたコンピュータおよびコンピュータ技術でできるだけ高速なプログラム実行を達成するためには、できるだけ効率的に実現されることが重要である。

【0006】サブプログラムの呼び出しおよびそのサブプログラムからの復帰を実現する従来広く行われている方法は、CALLおよびRETURN文である。この高水準命令は、スタック（後入れ先出しメモリ構造）とともに作動する。実行を終了していない各サブプログラムはスタックに情報のフレームを有する。スタック・フレームには、引数すなわちパラメータ、局所変数、返値、およびサブプログラムに付随したその他の情報が格納されている。

【0007】CALL文によって、コール中のサブプログラムのコンテキストがスタック・フレームに格納され、コールされたサブプログラムにスタック・フレームが作成される。コンテキストは、汎用レジスタの内容、命令ポインタの内容、および条件コードのような情報を含む。RETURN文によって、リターンする、コールされていたサブプログラムのスタック・フレームがスタックから削除され、プロセッサは、リターン先の、コールしていたサブプログラムの格納コンテキストを復元する。

【0008】メモリ内容およびプロセッサ状態のこのようなすべての操作には時間がかかるため、システム・スループットを減ずる。さらに、サブプログラム・コールの長い連鎖によって、スタックはメモリのかかなりの領域を占有するため、他の用途に使用可能なメモリの量が減少する。このことは、明らかなメモリ制限の他に、メイン・メモリとのページのスワッピングのような活動の生じる頻度が増加することによってシステム・スループットを減ずることもある。

【0009】従来のサブプログラムのコール・リターン装置のこれらの欠点のため、限定された範囲であっても、これらの欠点を改善する方法が必要である。

【0010】

【発明が解決しようとする課題】1つの知られたアセンブラおよびリンク・エディタの組合せは、「leafproc」として知られる機能を使用しており、これによって、リンク・エディタは標準的なコール命令を分岐リンク（BAL）命令に変更する。BAL命令は、次の（実行ではなくコンパイル順序に関する）命令のアドレスを、汎用レジスタのようなスタック外ロケーションに格納して、BAL命令のオペランドによって指定された目的命令への通常の分岐動作を実行する。従ってスタックの状態は不変である。

14

【0011】また、復帰は、BAL命令が復帰アドレスを格納した汎用レジスタをオペランドとする分岐命令によって実現される。しかし、BAL命令は、アセンブリ言語で書かれたサブプログラムの呼び出しにおける使用に制限され、しかもそのサブプログラムは、自分自身を含めて、他のサブプログラムを呼び出しておらず、数個より多くの汎用レジスタを必要としないものに制限される。

【0012】UNIXオペレーティング・システムは、「setjmp」および「longjmp」として知られる2個のペアの機能を使用している。一方のペアはオペレーティング・システム自体のアルゴリズムとして実現され、もう一方のペアはオペレーティング・システムへのユーザ・インタフェースにおけるライブラリ・ファンクションとして実現されている。

【0013】オペレーティング・システム・カーネルのレベルでは、「setjmp」アルゴリズムはコンテキスト・スイッチを実行するが、セーブされたコンテキストをスタック上に格納する代わりに、それを新プロセスのメモリ領域（u領域）にセーブする。このメモリ領域は、そのプロセスのコンテキストのみによってアクセスされる必要のあるプロセス制御情報を含む。従って実行は旧プロセスのコンテキストで継続する。

【0014】カーネルがセーブしたコンテキストを再開したい場合は、「longjmp」アルゴリズムを使用し、これはu領域から、セーブされたコンテキストを復元する。この技術は、プロセスの概念を含むオペレーティング・システム・カーネルの実現に制限される。これは、オペレーティング・システム・カーネルにのみ使用可能なシステム・レベルの機能であり、アプリケーション・プログラマによる使用によってアクセス可能なものではない。

【0015】ユーザ・インタフェースのレベルでは、「setjmp」ファンクションはコンテキスト・スイッチを実行し、旧コンテキストをスタックにセーブするが、そのコンテキストへのポインタは所定のスタック外ロケーションに格納する。その後、「longjmp」ファンクションは、パラメータとして所定のスタック外ロケーションのアドレスを与えられると、格納されたコンテキストに復帰し、復元される。

【0016】「longjmp」ファンクションは従来のRETURN文の多くのオーバーヘッドを節約することが可能であるが、「longjmp」ファンクションは、CPU命令サイクルやスタック・メモリ消費のような従来のCALL文のオーバーヘッドを縮小することはない。しかも、longjmp命令を含むサブプログラムはすべて、setjmp命令を実行した親サブプログラム（サブプログラム呼び出しの連鎖における先行サブプログラム）を有していなければならない。

【0017】従って、「longjmp」ファンクショ

15

ンを使用するサブプログラムは、任意のコンテキストから制限なしに呼び出されることが可能なサブプログラムとは異なり、「正常に」呼び出されることができない。さらに、setjmpは、再帰的連鎖の親サブプログラムによって使用されることは可能だが、再帰的呼び出し連鎖（サブプログラム呼び出しの列であって、その列の最後の呼び出しが列の第1呼び出しを生じるサブプログラムを呼び出してループを形成するもの）の中で使用されることはできない。

【0018】最後に、ある知られたインタプリタは、
「限嗣再帰法」として知られる機能を利用している。自己参照再帰ファンクション内でのみ使用可能であり、再帰ファンクションが復帰前に実行する最後の作用が自分自身を呼び出す場合にのみ使用可能であるため、この機能は呼び出されたファンクションの反復のためのスタック・フレームの作成を抑え、ファンクションの呼び出し反復のフレームを再使用する。従来のCALLおよびRETURN文の列のオーバヘッドの多くを除去するためには有効ではあるが、「限嗣再帰」機能は自己参照再帰ファンクション呼び出しのみに制限されるため、限定された適用可能性および有用性しか有しない。

【0019】

【課題を解決するための手段】本発明は、従来技術の以上およびその他の欠点を解決することを目的とする。本発明によれば、例えば以下でPASS_CONTROLと呼ばれる装置によって、相異なるサブプログラムの呼び出しの全系列からの、その系列を開始したサブプログラムへの直接の復帰が、その系列内の中間呼び出しを実行したサブプログラムへの復帰を妨げることなく実行される。

【0020】本発明の第1目的によれば、複数の相異なるサブプログラムのサブプログラム呼び出しの系列を実行する方法および装置が開示される。この系列は第1呼び出しを開始し、続いて少なくとも1個の中間呼び出しを実行し、その後最終呼び出しで終了する。サブプログラムの実行中は、呼び出しサブプログラムによるサブプログラムの系列内の各呼び出しに応じて、呼び出しサブプログラムの実行は停止し、呼び出されたサブプログラムの実行が開始される。

【0021】さらに、系列内の第1呼び出しに応じて、第1呼び出しを行った呼び出しサブプログラムの実行が行われる命令へのポインタが、実行スタックに格納される。例えば、対照的に、命令ポインタは、系列内のその他の呼び出しに応じては格納されない。従って、系列内の最終呼び出しによって呼び出されたサブプログラムからの復帰に応じて、復帰されたサブプログラムの実行は停止し、格納された命令ポインタが実行スタックから回復され、系列内の第1呼び出しを行ったサブプログラムの実行が、回復された命令ポインタによって指示された命令から再開される。最後に呼び出されたサブプログラ

16

ムから、呼び出しの系列を開始したサブプログラムへのこの実行の分岐は直接であり、系列内の中間呼び出しを行うサブプログラムの実行の中間的な再開がない。

【0022】最後に呼び出されたサブプログラムからの復帰が、呼び出し系列を開始したサブプログラムへ直接実行されるため、各呼び出しサブプログラムと呼び出されたサブプログラムの間の個々の復帰を実行し、個々の復帰を実行するための実行スタックを操作するために通常消費された時間が節約される。結果として、システム・スループットおよび性能はモジュール化実現において改善される。

【0023】しかしながら、この方法および装置は、呼び出し系列およびそこから復帰を実行するために、いくつかの従来の装置で行われているように例外的な特別のスタック外機構を使用するのではなく、従来の実行スタックを使用する。従って、従来技術とは異なり、サブプログラムはあらゆる状況で正常に呼び出されることが可能である。すなわち、1つのプログラム内で、同一のサブプログラムが、標準的なCALL装置およびPASS_CONTROL装置のいずれによって呼び出されることも可能であり、実行システムは、使用された呼び出し型を追跡する必要も、その呼び出し型の結果によって動作を調節する必要もない。

【0024】系列内の呼び出しによって呼び出されるすべてのサブプログラムは実行コンテキストを共有することが望ましい。これによって、呼び出しおよび復帰の実行に消費される時間がさらに短縮される。系列内の第1呼び出しに応じて、第1呼び出しを実行した呼び出しサブプログラムの実行コンテキストが実行スタックに格納される。対照的に、実行コンテキストは、系列内のその他の呼び出しに対してはセーブされない。むしろ、存在するコンテキストが保持され共有される。また、系列内の最終呼び出しによって呼び出されたサブプログラムからの復帰に応じて、実行コンテキストはスタック上に格納されたコンテキストに還元される。もちろん、この復帰は、系列内の中間呼び出しを実行したサブプログラムのコンテキストの従来の中間的な還元なしに、直接的に実行される。

【0025】さらに、呼び出し系列中に実行スタックによって消費されるメモリの量は、呼び出された各サブプログラムに対して新たなスタック・フレームを作成するのではなく、呼び出し系列によって呼び出されるすべてのサブプログラムが単一のスタック・フレームを共有することによって、大幅に縮小され限定されることが望ましい。系列内の第1呼び出しに応じて、呼び出されたサブプログラムに対するスタック・フレームが実行スタックに作成され、呼び出しサブプログラムの命令ポインタおよび実行コンテキストがそこに格納される。

【0026】対照的に、系列内のその他の呼び出しに応じては、スタック・フレームは作成されず、命令ポイン

タおよび実行コンテキストもスタック・フレームに格納されない。呼び出された各サブプログラムに対する情報、例えば、局所および一時変数は、次の2つの方法のうちの1つによって共有スタック・フレームに格納される。1つの方法は、呼び出されたすべてのサブプログラムによって共有されるスタック・フレームの領域に格納する方法であり、もう1つの方法は、呼び出された各サブプログラムが拡張可能共有フレーム内に自分の使用のために確保した複数の領域のうちの1つに格納する方法である。

【0027】パラメータは、各呼び出しサブプログラムによって、呼び出された各サブプログラムへ、汎用レジスタを通して、または、呼び出し系列を開始したサブプログラムのスタック・フレームの共有パラメータ領域を通して渡される。後者の共有パラメータ領域は実行スタックの共有スタック・フレームに先行する。また、最後に呼び出されたサブプログラムからの復帰に応じて、実行コンテキストおよび命令ポインタは現在のスタック・フレーム（この場合は共有スタック・フレーム）から復元され、現在の全スタック・フレームは、すべての復帰呼び出しによって行われるように、実行スタックから削除される。

【0028】本発明の第2目的によれば、上記で特徴づけられる機能は、サブプログラム呼び出し文およびサブプログラム復帰文を含むソース・コード文からなる複数の相異なるソース・サブプログラムで構成されたソース・プログラムをコンパイルするための新しいコンパイラ装置によって実現される。特に、ソース・プログラムは、複数の相異なるソース・サブプログラムの呼び出しの文の系列を含む。

【0029】系列内の第1呼び出し文を除いて、系列内の文は、系列内の先行する呼び出し文によって呼び出されたソース・サブプログラムにそれぞれ含まれる。この系列は、第1呼び出し文で開始し、少なくとも1個の中間呼び出し文が続き、最終呼び出し文で終了する。コンパイラ装置はソース・プログラムをコンパイルして、オブジェクト命令からなる複数のオブジェクト・サブプログラムで構成されるオブジェクト・プログラムを生成する。

【0030】ソース・サブプログラム内で、ソース・サブプログラムを呼び出す文に遭遇すると、コンパイラ装置は呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行から、呼び出されたソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行へ分岐する命令を生成する。ソース・サブプログラム内で、系列内の第1呼び出しであるCALL文に遭遇すると、コンパイラ装置は、第1呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行を再開する命令へのポインタを実行スタック

に格納する命令を生成する。

【0031】例えば、対照的に、コンパイラ装置は、系列内の後続呼び出しであるPASS_CONTROL文に遭遇した場合には、命令ポインタを格納する命令を生成しない。また、系列内の最終呼び出し文によって呼び出されたソース・サブプログラム内の復帰文に遭遇すると、コンパイラ装置は、実行スタックから格納された命令ポインタを回復し、復帰文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行から、系列内の第1呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行へ、実行スタックから回復された命令ポインタによって指示された命令において分岐し、系列内の中間呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行を途中で再開することなく直接的に分岐する命令を生成する。コンパイラ装置はまた、上記で特徴づけられた付加的な機能を、対応するオブジェクト命令を生成することによって実現する。

20 【0032】

【実施例】本発明を理解し評価する簡単な方法は、従来のコール・リターン装置と比較することである。従って、以下の説明はこの装置のかなり詳細な説明から開始される。

【0033】図1は、格納プログラム制御下で動作する汎用データ・プロセッサの例（以下でコンピュータ100と呼ぶ）のブロック図である。コンピュータ100は、メモリ101および中央処理装置（CPU）102を含む。CPU102は通常、算術論理装置（ALU）606およびレジスタ・ファイル605を含む。レジスタ・ファイル605は、少なくとも、命令ポインタ（IP）レジスタ603、スタック・ポインタ（SP）レジスタ602、フレーム・ポインタ（FP）レジスタ601、および複数の汎用レジスタ604を含む。

【0034】ここでのメモリ101は、RAM、ROMやディスクのような任意の形式の記憶装置を表す。これは、CPU102によって実行されるプログラム111、および、その実行中にプログラム111によって使用される入力情報110を含む。CPU102によるプログラム111の実行に従い、メモリ101はまた、入力情報110を使用してプログラム111の実行によって生成された出力情報112を含む。従って、CPU102は式 $f(a) = b$ によって定義されるファンクション要素113を実現するとみなされることが可能である。ただし、 f は実行プログラム111によって実現されるファンクション、 a は入力情報110、 b は出力情報112である。

【0035】プログラム111がコンピュータ100によって実行されるためには、プログラムはコンピュータ100の論理（例えばALU606）によって理解され

19

る形式でなければならない。この形式は一般的にオブジェクト・プログラムと呼ばれている。しかし、ほとんどのプログラムは最初にオブジェクト形式で書かれるのではなく、高水準のソース・プログラム形式で書かれる。

【0036】従って、プログラムは、コンピュータ100によって実行される前に、ソースからオブジェクト形式に変換されなければならない。この変換は、コンパイラ・オブジェクト・プログラム（または略してコンパイラ）によって実行される。従って、最初に、実行プログラム111はコンパイラであり、入力情報110はアプリケーション・ソース・プログラムであり、出力情報112は、コンパイラによってアプリケーション・ソース・プログラムから生成されたアプリケーション・オブジェクト・プログラムである。その後、アプリケーション・オブジェクト・プログラムが実行プログラム111となり、アプリケーション入力情報110上で動作して、アプリケーション出力情報112を生成する。

【0037】実行中、プログラム111は、メモリ101内に実現された実行スタック114を使用して、少なくともいくつかの部分的な入力および出力情報を、他の一時的情報とともに一時的に格納する。図8を見れば、スタック114とは後入れ先出しメモリ構造である。これには次の2個のポインタが付随する。スタック・ポインタ（SP）602はスタック114上で次の空き記憶ロケーションを指示し、フレーム・ポインタ（FP）601は実行中のサブプログラムに付随したスタック部分の開始点すなわち最初の記憶ロケーションを指示する。このスタック部分はスタック・ポインタとフレーム・ポインタの間の範囲にあり、フレーム（例えばフレーム610、611、612）と呼ばれる。

【0038】本出願の焦点は、コンパイラによって生成されたオブジェクト・プログラムによるスタック114の効率的な使用法と、コンパイラがオブジェクト・プログラムによるこの効率的な使用を引き起こす方法である。

【0039】アプリケーション・ソース・プログラム110は、図2に示されたようなサブルーチン終端コールおよびリターンの連鎖を含むことがある。各サブプログラム200~209はその中に他のコールを含んでもよい。このコールは連鎖の一部とは見なされず、本出願の焦点の主題ではない。本出願は、呼び出されたサブプログラムの終端コール、すなわち、呼び出しサブプログラムへのリターンの前に呼び出されたサブプログラムによって実行される最後の文のみに関する。

【0040】図2に示されるように、この連鎖はメイン・サブプログラム200によるAサブプログラム201へのコールで始まる。Aサブプログラム201は次にBサブプログラム202をコールし、ただちにリターンで終了する。Bサブプログラム202は次にCサブプログラム203をコールし、ただちにリターンで終了する。

20

以下同様である。最後に、コールされたGサブプログラム208はHサブプログラム209をコールし、ただちにリターンで終了する。一方Hサブプログラム209は単にリターンで終了する。

【0041】Hアプリケーション209はコールしたGサブプログラム208にリターンし、続いてただちにそれをコールしたサブプログラムにリターンし、以下同様にして連鎖を上昇する。リターン先のCサブプログラム203はただちにBサブプログラム202にリターンし、これはただちにAサブプログラム201にリターンし、さらにこれはただちにメイン・サブプログラム200にリターンする。

【0042】上記で、サブプログラムの「終了」とは論理的終了である。すなわち、それはサブプログラムの実行中に実行される最後の文である。この「終了」は、物理的終了を意味しない。すなわち、この最後に実行される文がサブプログラムのリスティングの物理的に最後の文でなければならないことを意味する意図はない。

【0043】従来のコンパイラ111は、ソース・プログラム110内で、図2のようなCALLおよびRETURN文に遭遇すると、図3~5に示される方法で応答する。これらの図は一般的なコンパイラの論理的機能を示す。この機能は、コンパイラの特実の実現例の機能に正確に一致しないこともある。

【0044】図3を参照すれば、ステップ300で、CALL文に遭遇すると、ステップ301で、コンパイラ111は、コール側のサブプログラム（例えば、図2のメイン・サブプログラム200）によって被コール側のサブプログラム（例えば、図2のAサブプログラム201）に渡されたパラメータを計算するアプリケーション・オブジェクト・プログラム112命令を生成する。

【0045】次にステップ302で、コンパイラ111は一般的に、このパラメータをスタック114の領域701にプッシュし、対応してスタック・ポインタ602をインクリメントする命令を生成する。しかし、あるプロセッサでは、サブプログラム間のパラメータの受け渡しはスタック114を通してではなく大域レジスタ（図示せず）を通して行われる。このようなプロセッサの場合、コンパイラ111は、図3の破線で示されるようにステップ302を省略する。さらに、実際は、ステップ301および302はパラメータごとに混合することができる。

【0046】次に、コンパイラ111は、ステップ303で、フレーム・ポインタ601の値をスタック114の領域702にプッシュする命令を生成する。ステップ304で、フレーム・ポインタ601をスタック・ポインタ602と等置し、これによってフレーム・ポインタ601を次のフレームの開始点を指示するように移動する命令を生成する。ステップ305で、ステップ303を説明するためにスタック・ポインタ602をインクリ

21

メントする命令を生成する。次にコンパイラ111は、ステップ306で、実行されるべき次のオブジェクト・コード命令を指示する命令ポインタ603をスタック114の領域703にプッシュし、このスタック114エントリを説明するためにスタック・ポインタ602をインクリメントする命令を生成する。

【0047】次に、あるコンパイラ111は、ステップ307で、コール側のサブプログラムのコンテキストをスタック114の領域704にプッシュし、このエントリを説明するためにスタック・ポインタ602をインクリメントする命令を生成する。コンテキストとは、例えば汎用レジスタ604の内容のように、コンピュータ100がコール側のサブプログラムの実行を再開するために必要な情報である。しかし、ほとんどのコンパイラは、メモリ最適化として、ステップ307と同等のステップを異なるロケーションで実行する。従って、ステップ307は図3において破線で示されている。

【0048】スタック114に情報を格納する命令の生成を完了すると、コンパイラ111は、ステップ308で、被コール側のサブプログラムに分岐する命令を生成する。

【0049】この時点まで、アプリケーション・オブジェクト・プログラム命令のコンパイラ111による生成順序は実行順序に従っている。実行プログラム111になった場合のアプリケーション・オブジェクト・プログラムの実行順序は図3～5に点線で示されている。ステップ308で生成された分岐命令によって、アプリケーション・オブジェクト・プログラムの実行は図4の被コール側のサブプログラムとともに継続し、被コール側のサブプログラムおよびそれが呼び出した全サブプログラムの実行の終了後にのみコールしたサブプログラムにリターンする。

【0050】リターン後、コールしたサブプログラムによってコールされたサブプログラムにステップ302で渡されたパラメータは不要となるため、コンパイラ111は次にステップ309でスタック・ポインタ602をデクリメントすることによって対応するパラメータをスタック114からポップする命令を生成する。ステップ302とともに説明されたように、あるプロセッサは、スタック114を通してパラメータを渡さず、ステップ302は除去される。ステップ302がない場合、ステップ309も同様に存在しない。従って、ステップ309は図3の破線によってバイパスされて示されている。

【0051】この時点で、CALL文によって開始された命令生成は終了し、コンパイラ111は、ステップ310で、アプリケーション・ソース・プログラム110の次の文の処理に進む。同様に、アプリケーション・オブジェクト・プログラムが実行プログラム111になったとき、オブジェクト・コードの実行は、アプリケーション・ソース・プログラムの次の文から生じたオブジェ

22

クト・コード命令に進む。

【0052】図4を参照すると、ステップ400で、アプリケーション・ソース・プログラム110内のサブプログラム（例えば被コール側のサブプログラム）の始め、すなわち開始文に遭遇すると、一般的なコンパイラは、ステップ401で、上記のステップ307と同等な動作を実行する。しかし、コンパイラが、図3のステップ307を実行した種類のものである場合、コンパイラは図4のステップ401を実行しない。このことは図4でステップ401をバイパスする破線によって示されている。

【0053】次に、ステップ402で、コンパイラ111は、呼び出されたサブプログラムの局所および一時変数のためのスタック114上の領域700を確保するため、スタック・ポインタ602をインクリメントする命令を生成する。こうして、ステップ403で、準備は終了し、コンパイラ111は、呼び出されたサブプログラムの本体の処理に進む。

【0054】このように、アプリケーション・オブジェクト・プログラム111の実行は、図3のステップ308の分岐命令から、図4で生成された命令に進む。

【0055】図5を参照すると、ステップ501で、被コール側のサブプログラムにおいてRETURN文に遭遇すると、コンパイラ111は、ステップ502で、スタック・ポインタ602の値をフレーム・ポインタ601の値と等しく設定する命令を生成する。これは、スタック114からリターン元のサブプログラムの全フレームをポップする効果を有する。

【0056】次にステップ503で、コンパイラ111は、コール側のサブプログラムの実行の再開の準備をするため、スタック114から、リターン元のサブプログラムをコールしたサブプログラムのコンテキストにコンピュータ100を復元する命令を生成する。図3のステップ303～307を参照すると、コール側のサブプログラムのコンテキストは、スタック114上で、現在スタック・ポインタ602によって指示されているポップされたフレームの始点の2ワード後に位置する。

【0057】次に、ステップ504で、コンパイラ111は、スタック114から、格納されたフレーム・ポインタの値をフレーム・ポインタ・レジスタ601にロードする命令を生成する。図3のステップ303を参照すると、格納されたフレーム・ポインタの値は、現在スタック・ポインタ602によって指示されているポップされたフレームのワードにある。ステップ504で、フレーム・ポインタ601は、被コール側のサブプログラムのスタック・フレームの指示から、コール側のサブプログラムのスタック・フレームの指示にリターンする。

【0058】また、ステップ505で、コンパイラ111は、スタック114から、格納された命令ポインタの値を命令ポインタ・レジスタ603にロードする命令を

23

生成する。図3のステップ303～306を参照すると、格納された命令ポインタの値は、現在スタック・ポインタ602によって指示されているワードの次の、ポップされたフレームのワードにある。この時点で、RETURN文によって開始された命令生成は終了し、ステップ506で、コンパイラ111はアプリケーション・ソース・プログラム110の次の文の処理に進む。

【0059】図5において点線で示されているように、アプリケーション・オブジェクト・コードの実行は、オブジェクト・プログラムが実行プログラム111になったとき、ステップ505まで図5の命令生成の順序に従う。ステップ505の命令の実行に応じて、アプリケーション・オブジェクト・プログラムの実行は、命令ポインタ603によって指示される命令に分岐し、これによってコール側のサブプログラム、すなわち、図3のステップ309で生成された命令にリターンする。

【0060】図2に示されたCALLおよびRETURN文の連鎖に対するコンパイラ111の応答の最終結果は、実行プログラム111になった後に、図6から10に示された方法でメモリ101のスタック114を操作するアプリケーション・オブジェクト・プログラムである。メイン・サブプログラム200の実行中およびAサブプログラム201のコールの前には、スタック114は、図6に示されるように、メイン・サブプログラム200の局所および一時変数の領域700を含むメイン・サブプログラム200のフレーム610からなる。

【0061】Aサブプログラム201へのコールによって、フレーム610は、コールされたAサブプログラム201のパラメータの領域701を追加することによって拡張され、呼び出されたAサブプログラム201のための新フレーム611も作成される。新フレーム611は、図7に示されるように、先行フレーム610へのフレーム・ポインタを含む領域702と、コール側のメイン・サブプログラム200の命令内のリターン地点への命令ポインタを含む領域703と、コール側のメイン・サブプログラム200のコンテキストを含む領域704と、コールされたAサブプログラムの局所および一時変数のための領域700を有する。

【0062】同様に、図8に示されるように、Bサブプログラム202へのコールによって、コール側のAサブプログラム201のフレーム611が拡張され、コールされたBサブプログラムのための新フレーム612も作成される。この処理は継続し、図9に示されるように、Gサブプログラム208へのコールの時点で、スタック114は先行する各サブプログラムのフレームを含み、Gサブプログラム208へのコールによって、Gサブプログラム208のためのフレーム618がスタック114に追加される。最後に、図10に示されるように、Hサブプログラム209へのコールによってフレーム618は拡張され、Hサブプログラム209のためのフレー

24

ム619がスタック114に追加される。

【0063】Hサブプログラム209からGサブプログラム208へのリターンによってHサブプログラム209のフレーム619がスタック114から削除され、スタック114が図9に示される状態に還元される。対応する結果が、図2の連鎖の連続する各サブプログラムからのリターンに対して得られ、最後に、Cサブプログラム203からBサブプログラム202へのリターンによって、スタック114は図8に示された状態に還元され、Bサブプログラム202からAサブプログラム201へのリターンによってスタック114は図7に示される状態に還元され、Aサブプログラム201からメイン・サブプログラム200へのリターンによってスタック114は図6に示される状態に還元される。

【0064】上記の従来の動作と対照して、本発明によるコンピュータ100の動作は図11～20に示されるようになり、以下で説明される。

【0065】アプリケーション・ソース・プログラム110において、図2のサブルーチン終端CALLおよびRETURN文の連鎖は、図11に示されるPASS_CONTROL文の連鎖に置き換えられる。メイン・サブプログラム200は、この場合も、Aサブプログラム201へのコールによって連鎖を開始する。しかし、Aサブプログラム201は、Bサブプログラム202へバス・コントロールすることによって機能を終了し、Bサブプログラム202はCサブプログラム203へバス・コントロールすることによって機能を終了し、などとなる。最後に、Gサブプログラム208はHサブプログラム209へバス・コントロールすることによって機能を終了し、Hサブプログラム209は単にリターンによって終了する。

【0066】しかし、図1の従来の例とは異なり、Hサブプログラム209からのリターンはA201からG208までのサブプログラムの全連鎖をバイパスして、連鎖を開始したメイン・サブプログラム200に直接リターンする。Hサブプログラム209の実行が終了すると、中間でA201からG208までの他のサブプログラムの実行を再開することなく、メイン・サブプログラム200の実行はただちに再開される。従って、コール側から被コール側のサブプログラムへの一段ごとのリターンは削減されて単一のリターンにまとめられることにより、個々の一段ごとのリターンによって消費される処理時間が節約される。

【0067】上記のように、A201からG208までの呼び出された各サブプログラムは、次に呼び出されるファンクションへバス・コントロールすることによって機能を終了する。これは、サブプログラムは暗黙に終了することも、明示的に終了することもあるためである。明示的には、サブプログラムの最後に実行される文が新たなPASS_CONTROL文であって、これは、呼

25

び出されたサブプログラムへパス・コントロールし、呼び出し側のサブプログラムの終端を規定するために使用される。

【0068】暗黙には、A201からG208までのサブプログラムは、それぞれ、新たなPASS_CONTROL文の後に、従来のRETURN文で終了することがある。しかし、リターンはHサブプログラム209からメイン・サブプログラム200へ直接行われるため、A201からG208までのサブプログラムをRETURN文で終了することは、たとえそれが可能であっても、不要であり余分である。A201からG208までのサブプログラムにおいてPASS_CONTROLの後にRETURN文が含まれる場合、RETURN文に応じてコンパイルされたアプリケーション・オブジェクト・コード命令はバイパスされ、実行されることはない。

【0069】さらに、暗黙には、A201からG208までのサブプログラムは、従来のCALLおよびRETURN文で従来のように終了することも可能である。言い換えれば、アプリケーション・ソース・プログラムにおいて、図2のCALLおよびRETURN文の連鎖の置換は、例えばアプリケーション・プログラム110を書くときにプログラマによって明示的に行われる必要はなく、コンパイラ111自体によってアプリケーション・プログラム110のコンパイル中に暗黙に行われることが可能である。この例では、コンパイラ111は、サブプログラムの終端の文を検査して、CALLおよびRETURN文が実行されるサブプログラムの最後の2文であるかどうかを判定し、そうである場合には、それらをPASS_CONTROL文と等価なオブジェクト・コード命令に変換する。

【0070】呼び出されたサブプログラムの連鎖の終端からその連鎖の開始点へ直接単一のリターンを実行する上記の機能は、A201からH209までの呼び出されたサブプログラムによるスタック・フレームの共有によって容易にされ、実現される。言い換えれば、呼び出された各サブプログラムのための新たなスタック・フレームは作成されず、すべての呼び出されたサブプログラムに対して1個のスタック・フレームのみが作成され、順に使用される。

【0071】本発明によるコンパイラ111は、図12～14に示される方法で、ソース・プログラム110におけるPASS_CONTROLおよびRETURN文との明示的または暗黙の遭遇に対応する。コンパイラ111は、暗黙のPASS_CONTROL文でない従来のCALLおよびRETURN文に、上記の従来の方法で対応する。図3～5の場合のように、図12～14はコンパイラの論理的機能を示す。

【0072】図12を参照すれば、ステップ1200でCALL文に遭遇すると、コンパイラ111は、ステッ

26

プ1201でアプリケーション・ソース・プログラム110の次の文をチェックし、ステップ1202でそれがRETURN文であるかどうか判定することによって、前記CALL文が暗黙のPASS_CONTROL文であるかどうかチェックする。次の文がRETURN文でない場合、前記CALL文は暗黙のPASS_CONTROL文ではなく、コンパイラ111は図3に示された方法でそのCALL文を従来のように処理する。しかし、CALL文に続く文がRETURN文である場合、そのCALL文は暗黙のPASS_CONTROL文であり、コンパイラ111はステップ1211に進む。

【0073】また、ステップ1210で、明示的PASS_CONTROL文に遭遇すると、コンパイラ111は直接ステップ1211に進む。

【0074】ステップ1211で、コンパイラ111は、呼び出されたサブプログラムへ呼び出し側のサブプログラムによって渡されているパラメータを計算するアプリケーション・オブジェクト・プログラム112の命令を生成する。これは、図3のステップ301と同等である。次にコンパイラ111は、一般的に、パラメータをスタック114にプッシュする命令を生成する。

【0075】サブプログラムは、スタック上のそれ自身のフレームに先行するフレームにパラメータを発見することを予期する。しかし、この例では、呼び出されたサブプログラムに対して新たなスタック・フレームは作成されない。従って、パラメータは、先行するフレーム、すなわち、メイン・サブプログラム200のフレーム610のパラメータ領域に格納されなければならない。これを行うため、コンパイラ111は次のファンクションを実行する。

【0076】コンパイラ111は、ステップ1212で、スタック・ポインタ602の現在の値を一時的にセーブする命令を生成し、ステップ1213で、スタック・ポインタ602を、呼び出されたサブルーチンに渡されているパラメータの数をフレーム・ポインタ601から引いた値に設定する命令を生成する。これは、スタック・ポインタがメイン・サブルーチン200のスタック・フレーム610のパラメータ領域701を指示する効果を有する。

【0077】スタック・ポインタ602がパラメータ領域701の外側、すなわち、前方の場所を指示しないことを保証するため、パラメータの最大許容数に制限が課され、パラメータの許容数のための空間が図12のステップ1213においてスタック上で使用可能であることを保証するために、図3のステップ302で前もって生成された命令がその最大許容数だけスタック・ポインタ602をインクリメントしなければならない。これは、Aサブプログラム201を十分多くのダミー・パラメータとともにコールしてパラメータの総数が最大数に等しくなるようにするか、または、パラメータ領域として確

保される人きさを宣言することが可能な「pragma」文をサポートすることによって可能となる。

【0078】スタック・ポインタ602が先行スタック・フレーム610のパラメータ領域701を指示すると、コンパイラ111は、ステップ1214で、計算されたパラメータをスタック114上にプッシュし、対応してスタック・ポインタ602をインクリメントする命令を生成する。これは、図3のステップ302と同様である。次にステップ1215で、コンパイラ111は、ステップ1212でセーブされた値をスタック・ポインタ602に復元し、それによってスタック・ポインタ602が再びスタック114の「トップ」を指示するようにする命令を生成する。

【0079】上記では、コンパイラ111は一般的にパラメータをスタック上にプッシュする命令を生成とした。しかし、図3とともに説明されたように、あるプロセッサでは、パラメータはスタック114を通してではなく大域レジスタを通して渡される。この場合、コンパイラ111は、図12の破線で示されるように、ステップ1212～1215を省略する。

【0080】次に、ステップ1216で、コンパイラ111は、呼び出されたサブプログラムへ分岐する命令を生成する。以下で明らかになるように、呼び出されたサブプログラムはいくつかのエントリ・ポイントを有する。図3のステップ308の場合のように呼び出されたサブプログラムの開始点に分岐する命令を生成する代わりに、コンパイラ111は、呼び出されたサブプログラムにおいて、呼び出されたサブプログラムの開始点からある所定量（実現オプションに依存する）をオフセットとするポイントに分岐する命令を生成する。

【0081】この時点で、PASS_CONTROL文によって開始された命令生成は終了し、ステップ1217で、コンパイラ111はアプリケーション・ソース・プログラム110の次の文の処理に進む。この処理は従来のものでよい。

【0082】また、図12の破線および破線ボックスで示されるように、コンパイラ111は、ステップ1218で、アプリケーション・ソース・プログラムにおける次の文をチェックして、それがRETURN文であるかどうか判定することが可能である。上で説明されたように、明示的または暗黙のPASS_CONTROL文に続くRETURN文は余分であり、コンパイラ111は、それを無視して、ステップ1217に戻り、そのRETURN文に続く次のアプリケーション・ソース・コード文に進み、処理することが可能である。

【0083】しかし、ステップ1218で、暗黙または明示的PASS_CONTROL文に続く文がRETURN文でない場合、コンパイラ111は、ステップ1219で、従来のコンパイルを継続する。PASS_CONTROL文に続く文への到達可能性の判定基準はRE

TURN文に対するものと同一である。

【0084】ステップ1216まで、コンパイラ111によるアプリケーション・オブジェクト・プログラム命令112の生成順序は、実行順序に従う。実行プログラム111になったときのアプリケーション・オブジェクト・プログラムの実行順序は図12～14の点線で示される。ステップ1216で生成される分岐命令のため、アプリケーション・オブジェクト・プログラムの実行は図13の被コール側のサブプログラムに継続する。

【0085】図13は、生成されるアプリケーション・オブジェクト・プログラムが複数のエントリ・ポイントを有することを除いては、図4と機能的に等価である。従って、ステップ1300で、アプリケーション・ソース・プログラム110においてサブプログラム（例えば呼び出されたサブプログラム）の開始すなわちエントリ文に遭遇すると、コンパイラ111は、図4で説明されたように図4と同等のファンクションを実行する。

【0086】図13においてコンパイラ111によって生成されたアプリケーション・オブジェクト・サブプログラムの実行は図4で説明されたように進行する。すなわち、従来のCALL文の結果として従来のようにこのサブプログラムへ分岐すると、図3でステップ307が実行されていない場合、オブジェクト・サブプログラムの実行はステップ1301で生成されたコードで開始し、図3でステップ307が実行された場合は、ステップ1302で生成されたコードで開始する。これは、図4で説明されたことと直接対応する。

【0087】暗黙または明示的PASS_CONTROL文の結果として図13のオブジェクト・サブプログラムの実行に分岐した場合、呼び出し側のコンテキストをセーブする必要はない。従って、オブジェクト・サブプログラムの実行は、ステップ1301で生成された命令の後のポイント（もし存在すれば）から開始する。

【0088】さらに、スタック・フレームの共有に関する2つのオプションが本実施例において提供される。第1のオプションは、呼び出されたサブプログラムが共有スタック・フレーム1620の全領域700～704を共有することである。第2のオプションは、呼び出された各サブプログラムがそれ自身の局所および一時変数領域700を共有スタック・フレーム1620内に有することである。

【0089】第1オプションのもとで、コンパイラ111は、図12のステップ1216で、アプリケーション・オブジェクト・サブプログラムにおいてオブジェクト・サブプログラムの開始点からの固定オフセットであるポイントへ分岐する命令を生成し、ステップ1303で生成された命令で開始することにより、図13で生成される全命令を実効的にバイパスする。

【0090】第2オプションのもとで、コンパイラ111は、再び図12のステップ1216で、アプリケーシ

29

ョン・オブジェクト・サブプログラムにおいて図13のステップ1302で生成された命令で開始する固定オフセット・ポイントへ分岐する命令を生成する。これらのオフセット・ポイントは、コンパイラによって処理された全サブプログラムに対して固定されている、すなわち、同一である。

【0091】図14を参照すれば、ステップ1400で、呼び出されたサブプログラムにおいて、図12のステップ1218~1219に示される方法でオプションとして処理されないRETURN文に遭遇すると、コンパイラは、図5に示されるファンクションを実行することによって従来の方法で対応する。

【0092】アプリケーション・オブジェクト・プログラムが実行プログラム111になった場合、アプリケーション・オブジェクト・コードの実行中には、Hサブプログラム209のRETURN文に応じてコンパイラによって生成されたコードは、図11のA201からH209までのサブプログラムをコンパイルした結果として生成されたアプリケーション・オブジェクト・コードにおいて遭遇する最初で唯一のRETURNコードである。

【0093】Hサブプログラム209に対して図5のステップ502でコンパイラによって生成されたコードの実行により、メイン・サブプログラム200のフレーム610に続く全情報がスタック114からポップされる。ステップ503~505でコンパイラによって生成されたコードの実行により、コンピュータ100は、メイン・サブプログラム200がAサブルーチン201を呼び出したときの状態に復元される。従って、ステップ505で実行される分岐に応じて、実行は、メイン・サブプログラム200の、図3のステップ309で生成される命令に復帰する。この過程で、A201からG208までのサブプログラムのRETURN文に対して生成された可能性のあるオブジェクト・コードはスキップされ、実行されることはない。

【0094】図11に示される暗黙または明示的PASS_CONTROL文の連鎖に対するコンパイラ111の応答の最終結果は、実行プログラム111になった場合、図15~20に示される方法でメモリ101内のスタック114を操作するアプリケーション・オブジェクト・プログラム112である。図15および16に示されるように、メイン・サブプログラム200によるAサブプログラム201へのコールの前後で、スタック114は、従来技術に対して説明されそれぞれ図6および7で示されたものと同一である。しかし、図16で、Aサブプログラム201のスタック・フレームは以下の新たなファンクションを示すために「共有フレーム1620」と表示されている。

【0095】図17に示されるように、Aサブプログラム201がBサブプログラム202へパス・コントロー

30

ルした後、メイン・サブプログラム200のフレーム610のパラメータ領域701は被コール側のAサブプログラム201のパラメータを格納しておらず、呼び出されたBサブプログラム202のパラメータを格納している。

【0096】図13のステップ1302とともに説明されたオプション1によれば、共有フレーム1620の局所および一時変数領域700もまた被コール側のAサブプログラム201の変数を格納しておらず、呼び出されたBサブプログラム202の変数を格納している。しかし、オプション2によれば、共有フレーム1620の最初の局所および一時変数領域700は被コール側のAサブプログラム201の変数を格納し続け、共有フレーム1620は、呼び出されたBサブプログラム202の変数を格納するための追加の局所および一時変数領域700によって拡張される。

【0097】スタック114に対し、遭遇するすべてのPASS_CONTROL文について、対応する変更が行われる。従って、Gサブプログラム208の呼び出し後、図18に示されるように、フレーム610のパラメータ領域701はGサブプログラム208のパラメータを格納し、共有フレーム1620の単一の局所および一時変数領域700はGサブプログラム208の変数を格納しているか、または、共有フレーム1620はA201からG208までの各サブプログラムに対して1個ずつ存在する複数の領域700を含むように拡張されている。

【0098】図19に示されるように、Hサブプログラム209の呼び出し後、フレーム610のパラメータ領域701はHサブプログラム209のパラメータを格納し、共有フレーム1620の単一の局所および一時変数領域700はHサブプログラム209の変数を格納しているか、または、共有フレーム1620はA201からH209までの各サブプログラムに対して1個ずつ存在する複数の領域700を含むように拡張されている。

【0099】図20に示されるように、Hサブプログラム209からのリターンに従って、スタック114は直接、ただちに、Aサブプログラム201へのコールに先行する図15に示される状態に復元される。

【0100】もちろん、本発明の上記の実施例に対するさまざまな変更および修正が可能である。例えば、パラメータ領域は、再利用または共有を可能にするのに十分な大きさである限り、スタック上のどこに位置するかは重要でない。同じことが局所および一時変数領域についても成り立つ。

【0101】

【発明の効果】以上述べたごとく、本発明によれば、PASS_CONTROLと呼ばれる装置によって、相異なるサブプログラムの呼び出しの全系列からの、その系列を開始したサブプログラムへの直接の復帰が、その系

31

列内の中間呼び出しを実行したサブプログラムへの復帰を妨げることなく実行される。

【図面の簡単な説明】

【図1】本発明の実施例の環境として使用されるコンピュータの機能ブロック図である。

【図2】図1のコンピュータの従来のCALL-RETURN呼び出し系列の機能図である。

【図3】図2の文を含むプログラムをコンパイルするときの従来のコンパイラの動作、および、コンパイルされた命令の図1のコンピュータによる実行のフロー図である。

【図4】図2の文を含むプログラムをコンパイルするときの従来のコンパイラの動作、および、コンパイルされた命令の図1のコンピュータによる実行のフロー図である。

【図5】図2の文を含むプログラムをコンパイルするときの従来のコンパイラの動作、および、コンパイルされた命令の図1のコンピュータによる実行のフロー図である。

【図6】図1のコンピュータによって、図3～5のコンパイルされた命令により実行される従来の実行スタック操作のブロック図である。

【図7】図1のコンピュータによって、図3～5のコンパイルされた命令により実行される従来の実行スタック操作のブロック図である。

【図8】図1のコンピュータによって、図3～5のコンパイルされた命令により実行される従来の実行スタック操作のブロック図である。

【図9】図1のコンピュータによって、図3～5のコンパイルされた命令により実行される従来の実行スタック操作のブロック図である。

【図10】図1のコンピュータによって、図3～5のコンパイルされた命令により実行される従来の実行スタック操作のブロック図である。

【図11】図1のコンピュータのPASS-CONTROL-RETURN呼び出し系列の機能図である。

【図12】図11の文を含むプログラムをコンパイルするときのコンパイラの動作、および、コンパイルされた命令の図1のコンピュータによる実行のフロー図である。

32

【図13】図11の文を含むプログラムをコンパイルするときのコンパイラの動作、および、コンパイルされた命令の図1のコンピュータによる実行のフロー図である。

【図14】図11の文を含むプログラムをコンパイルするときのコンパイラの動作、および、コンパイルされた命令の図1のコンピュータによる実行のフロー図である。

【図15】図1のコンピュータによって、図12～14のコンパイルされた命令により実行される実行スタック操作のブロック図である。

【図16】図1のコンピュータによって、図12～14のコンパイルされた命令により実行される実行スタック操作のブロック図である。

【図17】図1のコンピュータによって、図12～14のコンパイルされた命令により実行される実行スタック操作のブロック図である。

【図18】図1のコンピュータによって、図12～14のコンパイルされた命令により実行される実行スタック操作のブロック図である。

【図19】図1のコンピュータによって、図12～14のコンパイルされた命令により実行される実行スタック操作のブロック図である。

【図20】図1のコンピュータによって、図12～14のコンパイルされた命令により実行される実行スタック操作のブロック図である。

【符号の説明】

100 コンピュータ

101 メモリ

102 CPU

110 入力情報

111 実行プログラム

112 出力情報

114 実行スタック

601 フレーム・ポインタ・レジスタ

602 スタック・ポインタ・レジスタ

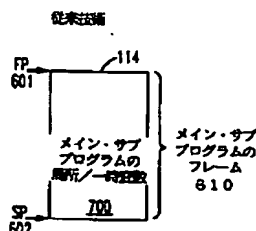
603 命令ポインタ・レジスタ

604 汎用レジスタ

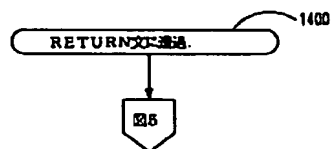
605 レジスタ・ファイル

40 606 ALU

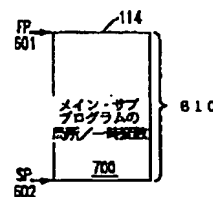
【図6】



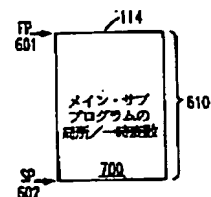
【図14】



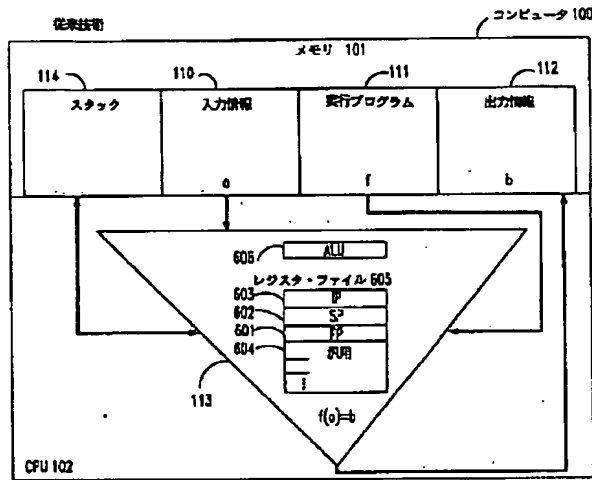
【図15】



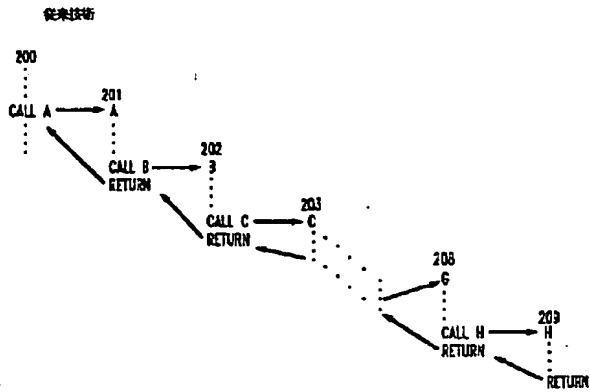
【図20】



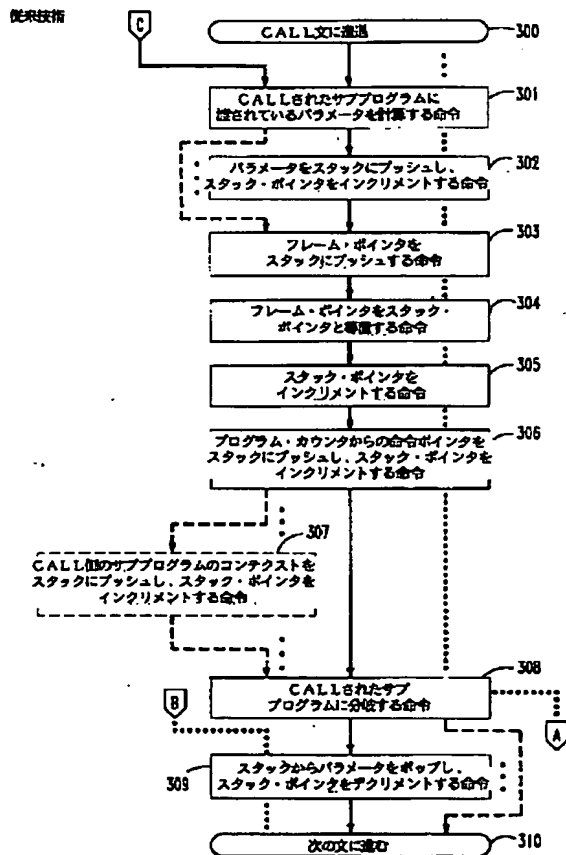
【図1】



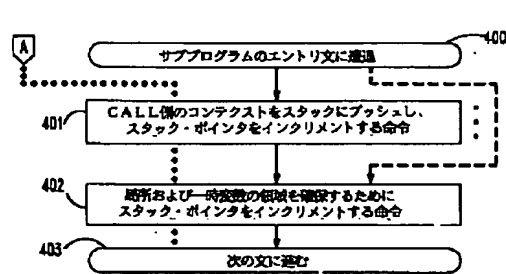
【図2】



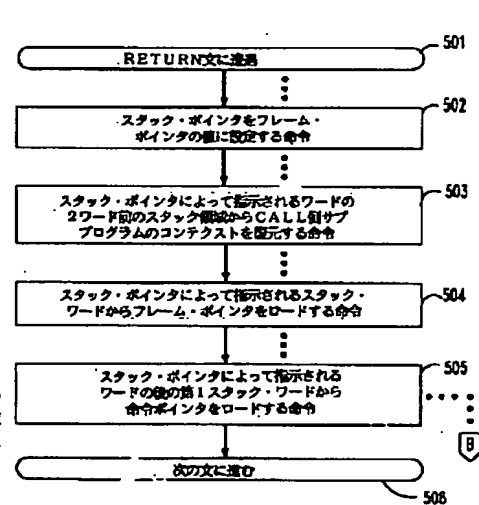
【図3】



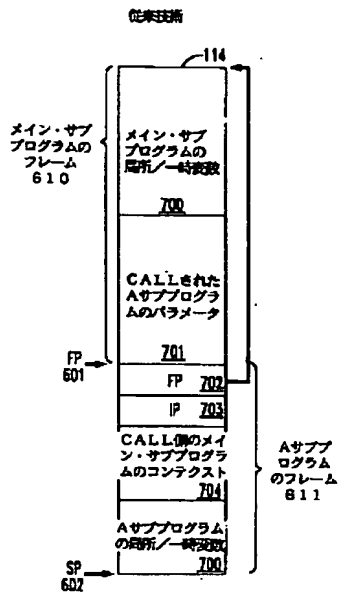
【図4】



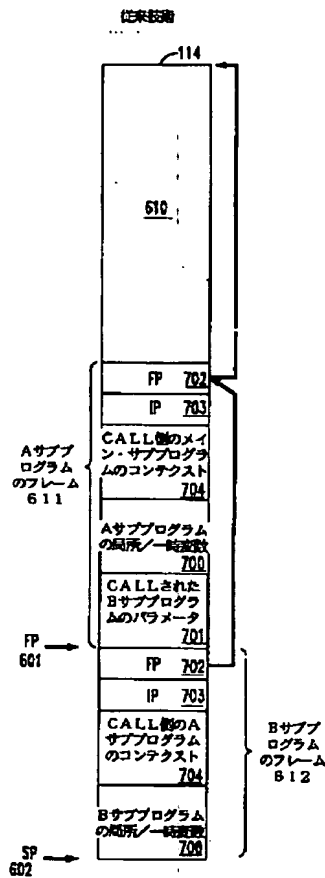
【図5】



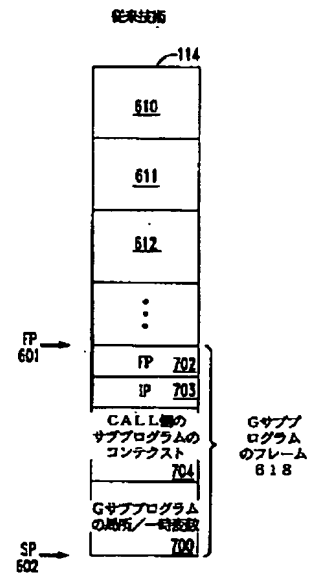
【図7】



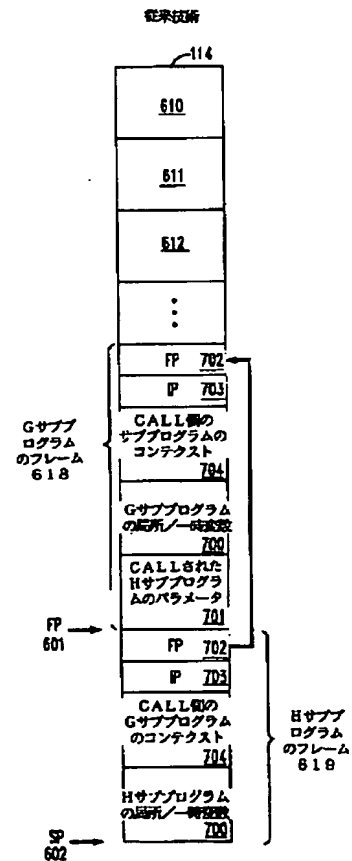
【図8】



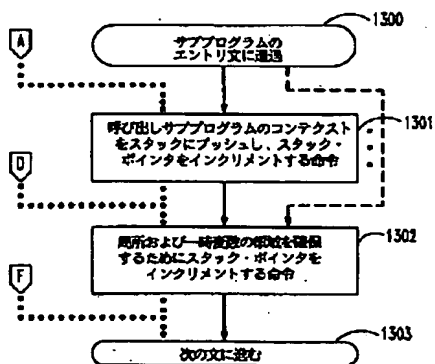
【図9】



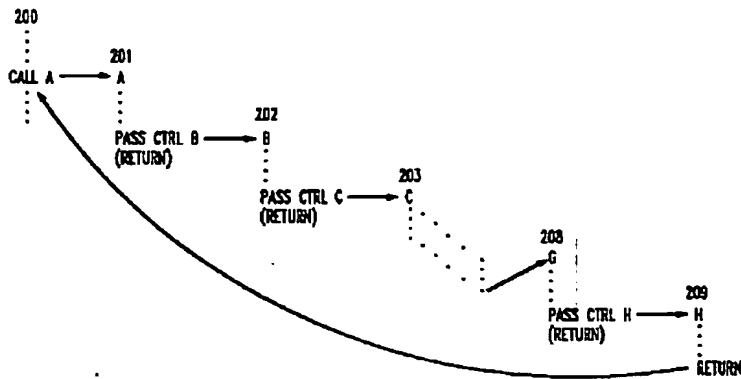
【図10】



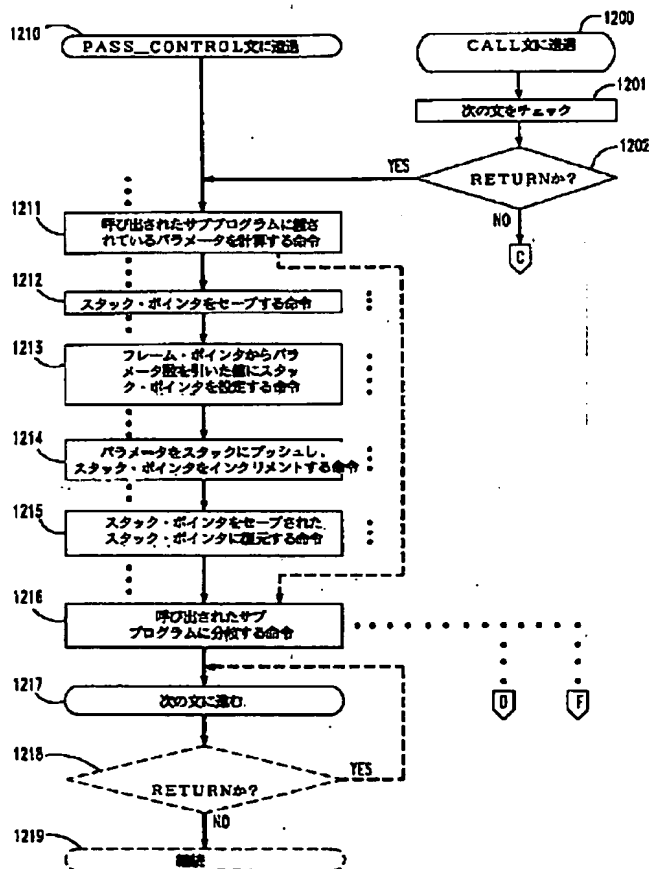
【図13】



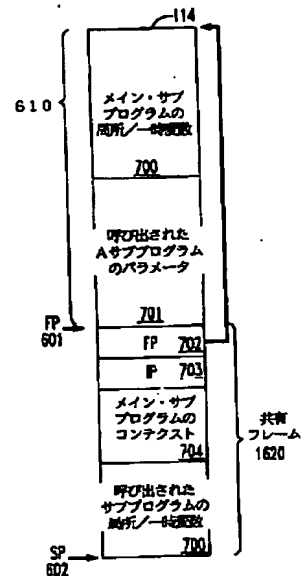
【図11】



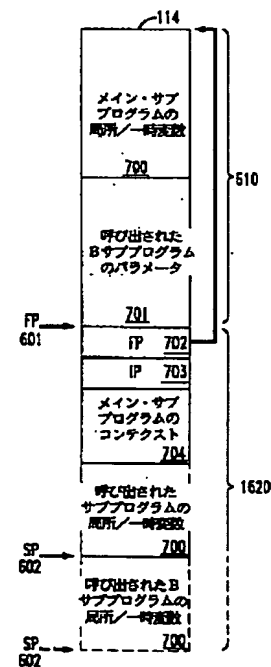
【図12】



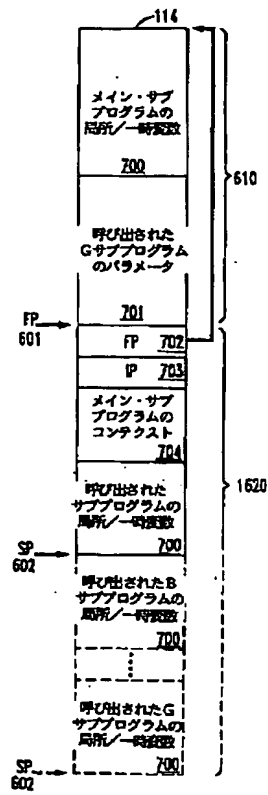
【図16】



【図17】



【図18】



【図19】

